UM EECS 270 F22
Introduction to Logic Design

21. State Assignment and Minimization

之前

is very difficult to find the best state assignment (because you have to try too many cases) and design the circuit all the way to the end to figure out if the cost of that assignment is reasonable

After you make the state diagram, state table, (capture the behavior) you need to do state assignment. You need to know how many flip flops you need, and what patterns to assign to them for various state.

is a very formal procedure that will give you actually the minimum number of states that you need.
→ creat many state as you need, and then you shove your state table into the state minimization procedure and find that you don't need that many states and you can get the same behavior with fewer state.

State Assignment

- Back to our combination lock example…

state table

S	X Y = 00	01	10	11	UNLOCK
A	A	B	A	A	0
B	A	B	C	A	0
C	A	D	A	A	0
D	A	B	C	A	1

S^+

one way to assign bits to the state

S	Q_1	Q_0
A	0	0
B	0	1
C	1	1
D	1	0

transition table

Q_1Q_0	X Y = 00	01	10	11	UNLOCK
0 0	00	01	00	00	0
0 1	00	01	11	00	0
1 1	00	10	00	00	0
1 0	00	01	11	00	1

Q_1^+ Q_0^+

We are doing D flip flops

So: $D_1 = Q_1^+$ $D_0 = Q_0^+$

What we do here:

seperately minimize for D_1 and D_0.

they will not share with the same gate.

Cost here:

How many literal? / How many inputs to the "OR" gate and "AND" gate.

$Q1^+$

Q_1Q_0 \ XY	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	0	1	0	0
10	0	0	0	1

4, 4, 4 → 12

$Q0^+$

Q_1Q_0 \ XY	00	01	11	10
00	0	1	0	0
01	0	1	0	1
11	0	0	0	0
10	0	1	0	1

3, 3, 4, 4 → 14

Minimal SOP: 26 literals 12+14 = 26

Minimal POS: 20 literals

Could it be less cost?

Perhaps we can do better using smarter state assignments…

- Another state assignment approach
  - Maximize the number of 1's

S	X Y 00	01	10	11	UNLOCK
A	A	B	A	A	0
B	A	B	C	A	0
C	A	D	A	A	0
D	A	B	C	A	1

notice that there are many S^+ A in the next state

① let A be "11": because that will increase the "1" in k map. ⟶ more opportunities for grouping ⟶ more grouping means lower cost.

② B occurs more often than C. But "01" and "10" have same number of "1". So just let B: 01 and C: 10.

③ D only appears once, so let D = 00.

S	Q_1	Q_0
A	1	1
B	0	1
C	1	0
D	0	0

Q_1Q_0	X Y 00	01	10	11	UNLOCK
1 1	11	01	11	11	0
0 1	11	01	10	11	0
1 0	11	00	11	11	0
0 0	11	01	10	11	1
			Q_1^+	Q_0^+	

$Q0^+$

Q_1Q_0 \ XY	00	01	11	10
00	1	1	1	0
01	1	1	1	0
11	1	1	1	1
10	1	0	1	1

$Q1^+$

Q_1Q_0 \ XY	00	01	11	10
00	1	0	1	1
01	1	0	1	1
11	1	0	1	1
10	1	0	1	1

Minimal SOP: 10 literals 2+2+2+2+1+1=10

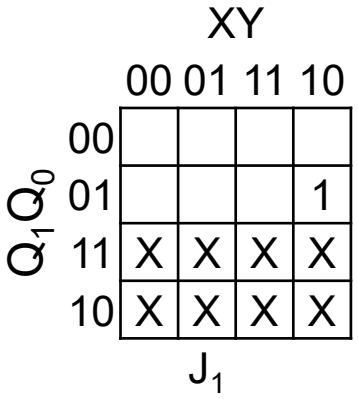
Minimal POS: 9 literals 3+4+2=9

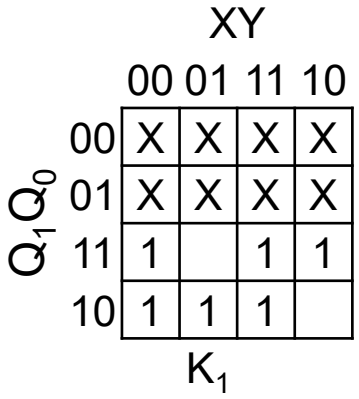
Using smarter state assignments improved the next-state circuit cost from 20 literals to 9 literals!

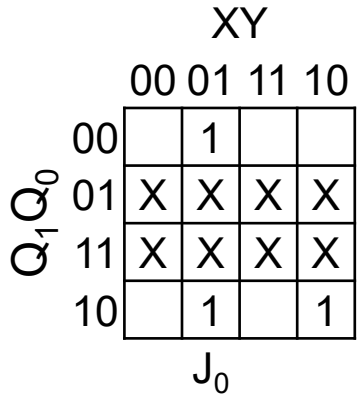
JK flip flops give you extra degree of flexibility with the don't care

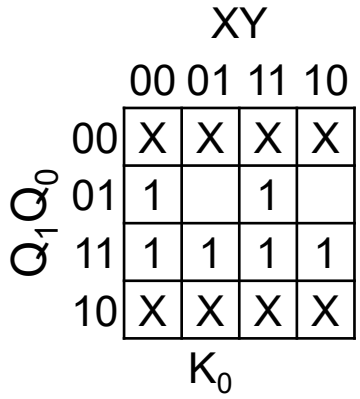
Combination Lock with JK FFs

	XY				XY				XY			
Q_1Q_0	00	01	11	10	00	01	11	10	00	01	11	10
00	00	01	00	00	0X	0X	0X	0X	0X	1X	0X	0X
01	00	01	00	11	0X	0X	0X	1X	X1	X0	X1	X0
11	00	10	00	00	X1	X0	X1	X1	X1	X1	X1	X1
10	00	01	00	11	X1	X1	X1	X0	0X	1X	0X	1X
		$Q_1^+Q_0^+$				J_1K_1				J_0K_0		









Minimal SOP: 19 literals

Minimal POS: 21 literals

Combination Lock with JK FFs Maximize 1s

S	X Y: 00	01	11	10	UNLOCK
A	A	B	A	A	0
B	A	B	A	C	0
C	A	D	A	A	0
D	A	B	A	C	1

S^+

S	Q_1	Q_0
A	1	1
B	0	1
C	1	0
D	0	0

Q_1Q_0	X Y: 00	01	11	10	UNLOCK
1 1	11	01	11	11	0
0 1	11	01	11	10	0
1 0	11	00	11	11	0
0 0	11	01	11	10	1

$Q_1^+ Q_0^+$

Q_1Q_0	XY: 00	01	11	10	XY: 00	01	11	10	XY: 00	01	11	10
00	11	01	11	10	1X	0X	1X	1X	1X	1X	1X	0X
01	11	01	11	10	1X	0X	1X	1X	X0	X0	X0	X1
11	11	01	11	11	X0	X1	X0	X0	X0	X0	X0	X0
10	11	00	11	11	X0	X1	X0	X0	1X	0X	1X	1X

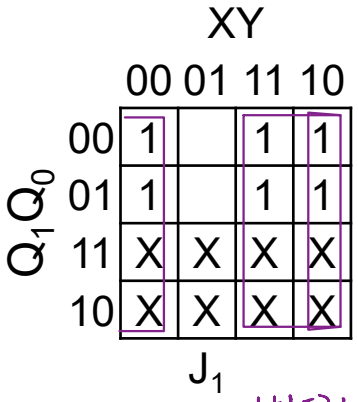
$Q_1^+Q_0^+$ J_1K_1 J_0K_0

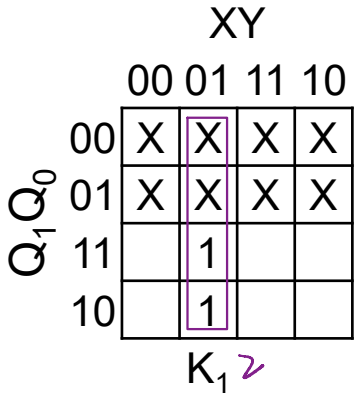
QQ^+	JK
00	0X
01	1X
10	X1
11	X0

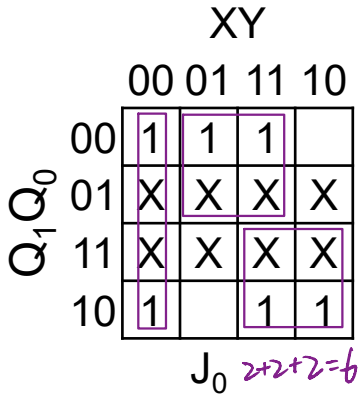
Excitation (required-input) Table

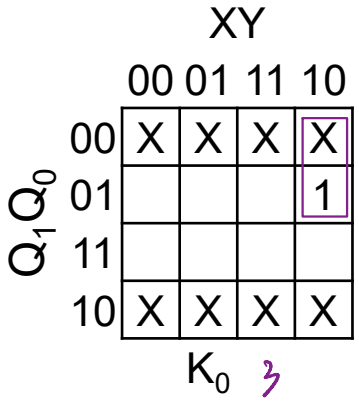
Combination Lock with JK FFs
Maximize 1s

	XY				XY				XY			
Q_1Q_0	00	01	11	10	00	01	11	10	00	01	11	10
00	11	01	11	10	1X	0X	1X	1X	1X	1X	1X	0X
01	11	01	11	10	1X	0X	1X	1X	X0	X0	X0	X1
11	11	01	11	11	X0	X1	X0	X0	X0	X0	X0	X0
10	11	00	11	11	X0	X1	X0	X0	1X	0X	1X	1X
		$Q_1^+Q_0^+$				J_1K_1				J_0K_0		









Minimal SOP: 13 literals 2+2+6+3=13.

Minimal POS: 13 literals

- Another approach: use more flip-flops
  - one-hot encodings (with the addition of 000)

	X Y				
S	00	01	10	11	UNLOCK
A	A	B	A	A	0
B	A	B	C	A	0
C	A	D	A	A	0
D	A	B	C	A	1
			S^+		

S	Q_2	Q_1	Q_0
A	0	0	0
B	0	0	1
C	0	1	0
D	1	0	0

		X Y			
$Q_2Q_1Q_0$	00	01	10	11	UNLOCK
0 0 0	000	001	000	000	0
0 0 1	000	001	010	000	0
0 1 0	000	100	000	000	0
1 0 0	000	001	010	000	1
		$Q_2^+ Q_1^+ Q_0^+$			

Read minterms directly off of transition table:

$$Q_0^+ = \overline{X}Y\left(\overline{Q}_2\overline{Q}_1\overline{Q}_0 + \overline{Q}_2\overline{Q}_1Q_0 + Q_2\overline{Q}_1\overline{Q}_0\right)$$
$$= \overline{X}Y\left(\overline{Q}_2\overline{Q}_1 + \overline{Q}_1\overline{Q}_0\right)$$
$$= \overline{X}Y\overline{Q}_2\overline{Q}_1 + \overline{X}Y\overline{Q}_1\overline{Q}_0$$

$$Q_1^+ = X\overline{Y}\overline{Q}_2\overline{Q}_1Q_0 + X\overline{Y}Q_2\overline{Q}_1\overline{Q}_0$$

$$Q_2^+ = \overline{X}Y\overline{Q}_2Q_1\overline{Q}_0$$

23 literals

How many states are *really* in our new state machine?

What happened to the other 4 states???

this method is safe: Because, the zero state is one of the good states. (state A). If in any unused states (u_1, u_2, u_3, u_4), your next state will be A

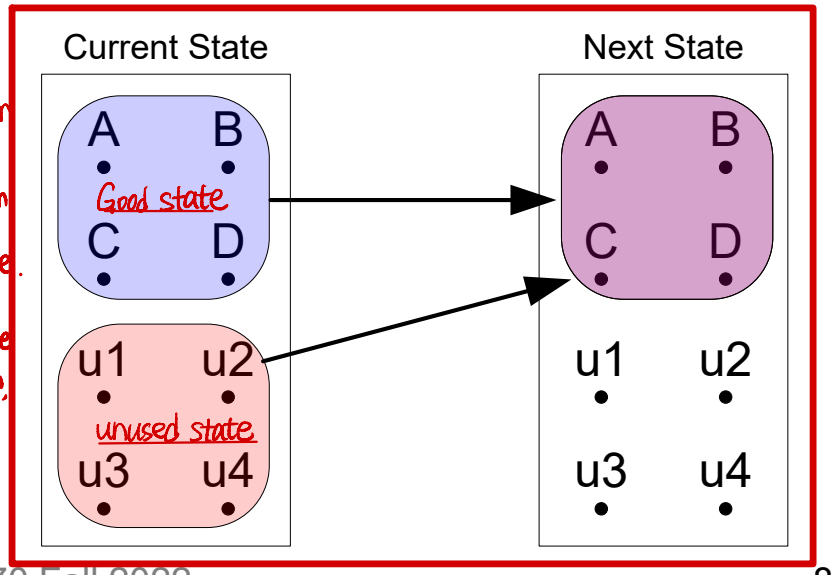
Unused States

- Previous design: *all unused states were implicitly assigned a next state of 000 (state A)*
- This is known as a **safe design**
  - If noise causes the machine to enter an unused state, it will return to a used state under any input conditions

				X Y				
	Q_2	Q_1	Q_0	00	01	10	11	UNLOCK
	0	0	0	000	001	000	000	0
	0	0	1	000	001	010	000	0
	0	1	0	000	100	000	000	0
	1	0	0	000	001	010	000	1
u1	0	1	1	000	000	000	000	0
u2	1	0	1	000	000	000	000	0
u3	1	1	0	000	000	000	000	0
u4	1	1	1	000	000	000	000	0

$\overline{Q_2^+\ Q_1^+\ Q_0^+}$

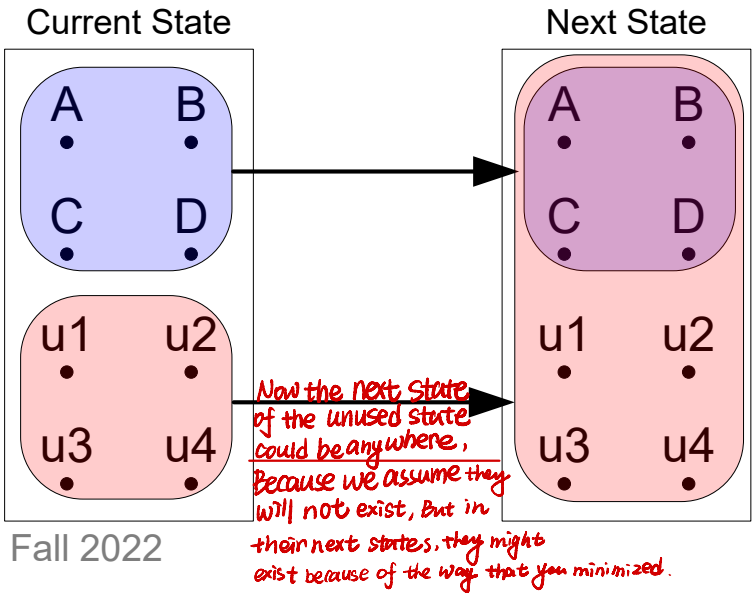
this is called safe design:
- Because, whenever you get into a bad state, your next state will be a good state.
- Also, these bad states are unreachable under normal operation.



- **Efficient Design**: Treat the next-states and outputs of unused states as don't cares
  - Minimizes circuit cost!
- If an unused state is ever entered, state machine may never return to normal operation

				X Y				
	Q_2	Q_1	Q_0	00	01	10	11	UNLOCK
	0	0	0	000	001	000	000	0
	0	0	1	000	001	010	000	0
	0	1	0	000	100	000	000	0
	1	0	0	000	001	010	000	1
u1	0	1	1	ddd	ddd	ddd	ddd	d
u2	1	0	1	ddd	ddd	ddd	ddd	d
u3	1	1	0	ddd	ddd	ddd	ddd	d
u4	1	1	1	ddd	ddd	ddd	ddd	d
				Q_2^+	Q_1^+	Q_0^+		

Finding transition equations now requires 5-variable K-maps!



- **State clustering** assigns unused states to behave like used states
- If noise causes an unused state to be entered, the machine will return to a used state in a single clock cycle

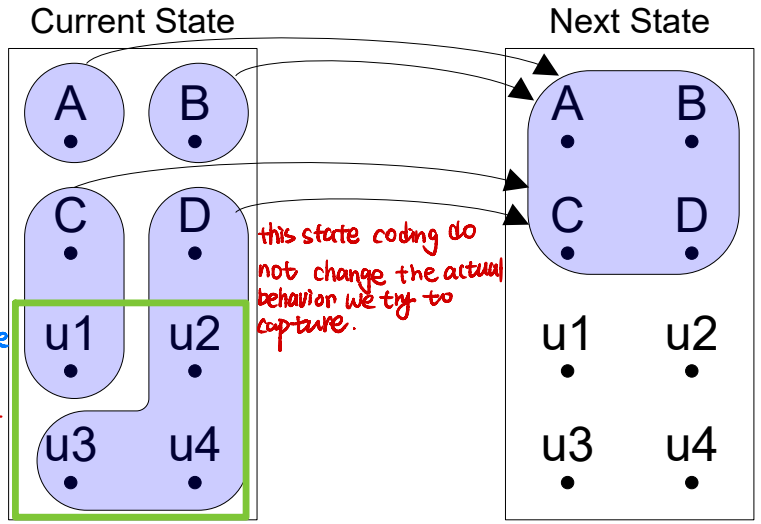
$Q_2Q_1Q_0$	X Y 00	01	10	11	UNLOCK
0 0 0	000	001	000	000	0
0 0 1	000	001	010	000	0
0 1 x	000	100	000	000	0
1 x x	000	001	010	000	1
	$Q_2^+ Q_1^+ Q_0^+$				

let "010" & "011" represent the same state

Represents 010 (C) and 011 (u1) one bad state prepares with the good state.

Represents 100 (D), 101 (u2), 110 (u3), and 111 (u4)

good bad



this state coding do not change the actual behavior we try to capture.

What we do here:

take each of my bad states (unused states), and make it a sibling to a good state

meaning: I will have 2 codes or 4 codes for the good states.

The cost of the design:

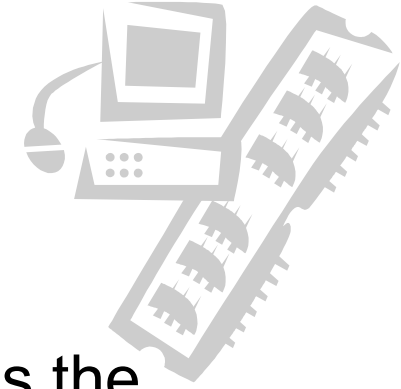
the total number of gates in the entire design → gates in the flip flops.
may be possible some times to have more flip flops (meaning more gates there). But fewer gates that drive the flip flops, 但不能确保这样, 即使是这个情况. → gates in the logic that drives the flip flops.

unreachable state:

because, there is no way under normal operation to go from from one of the state that you start from to go into them.

But they exist, so you must have some way of making sure that if you do enter one of them, then you have a way to get back to good state.

State Minimization



- Goal: Design a sequential circuit that satisfies the requirements of the given problem description (spec.)
- Follow sequential analysis steps in reverse! (more or less)
  - (optional) Construct state diagram
  - Construct state/output table

"Art of design"

  - Create state assignments
  - Create transition/output table
  - Choose FF type
  - Construct **excitation/output table**
    - Similar to transition/output table
  - Find excitation and output logic equations

"Turn the crank"

Sequential Design Review

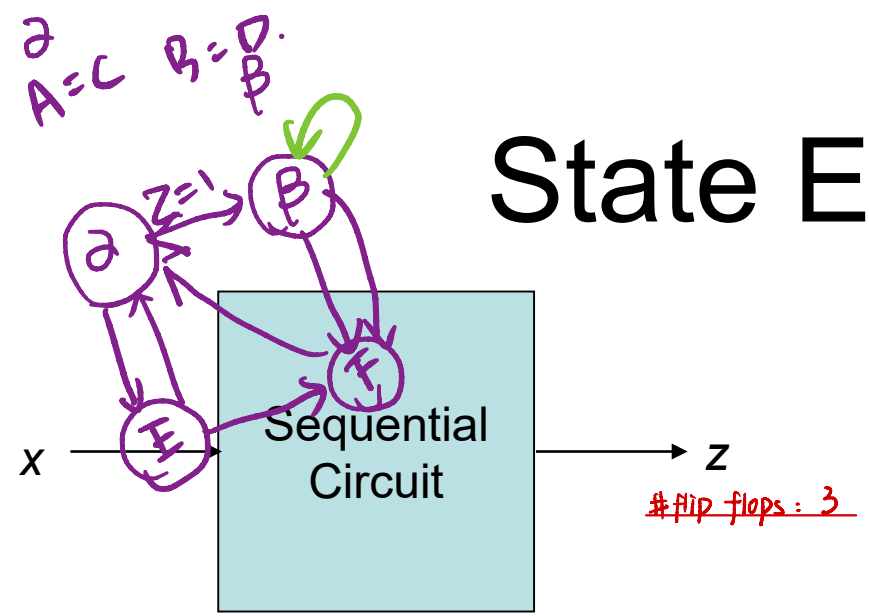
- Goal: Design a sequential circuit that satisfies the requirements of the given problem description (spec.)
- Follow sequential analysis steps in reverse! (more or less)
  - (optional) Construct state diagram
  - Construct state/output table

"Art of design"

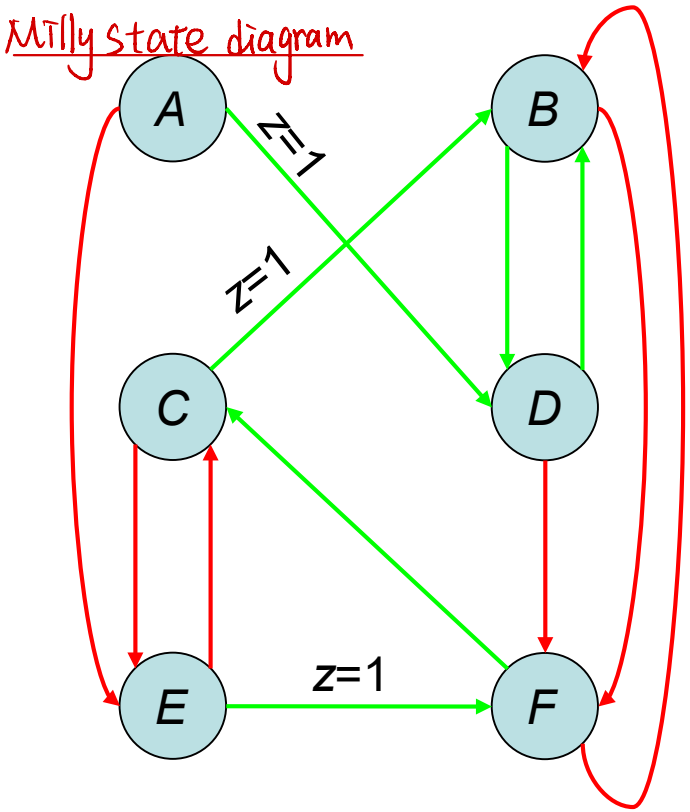
  - Minimize number of states
  - Create state assignments
  - Create transition/output table
  - Choose FF type
  - Construct **excitation/output table**
    - Similar to transition/output table
  - Find excitation and output logic equations

"Turn the crank"

State Equivalence



- States *S* and *T* are **equivalent** (aka **indistinguishable**) iff applying the same input sequence $x_1\ x_2\ \ldots$ yields the same output sequence $z_1\ z_2$ when starting in either state *S* or state *T*
- States *S* and *T* cannot be distinguished from the "outside." On the "inside" the circuit is redundant
- Equivalent states can be **merged**
- We need only consider sequences of length *n*-1 where *n* is the number of states (why?)



answer:

when I need to stop? : if I apply 1000 steps, and I can't tell them apart, should I keep going?

↳ Because, state diagram is a graph, and the graph has the "diameter"

So if I have a state diagram has n state, what is the graph that has the largest diameter? → circule (binary counter) . the distance to get back the same state : n-1 steps . (max)

Detecting Equivalent States Merger Table

present state / next state

PS	NS, z (output)	
	input x = 0	x = 1
A	E 0	D 1
B	F,0	D,0
C	E,0	B 1
D	F,0	B,0
E	C 0	F 1
F	B,0	C,0

Merger Table procedure

判断方法: Start out 2 state and apply the same inputs and see if we get the same output.

Example:
start from A and B.
- apply x=0
output of A = output of B = 0.
So we can't distinguish A and B by just apply x=0.
- apply x=1.
output of A ≠ output of B with the sequence of length "1"
I can tell A from B.
So A and B can not be equivalent

means: at the end if we can show B D are equivalent, then AC would be equivalent. ↑
How we deal with a longer sequence

Note: the state is always equivalent to itself
we don't need to check those
Note: A=B means B=A. So you only need to check through one direction
判断方法中判断: whether the pair is equivalent / not equivalent / potentially conditionally equivalent.

	A	B	C	D	E
A	AA				
B	X	BB			
C	BD	X	CC		
D	X	BD? ✓	X	DD	
E	CE DF	X	~~CE~~ BF	X	EE
F	X	~~FB~~ DC	X	BC BF	X

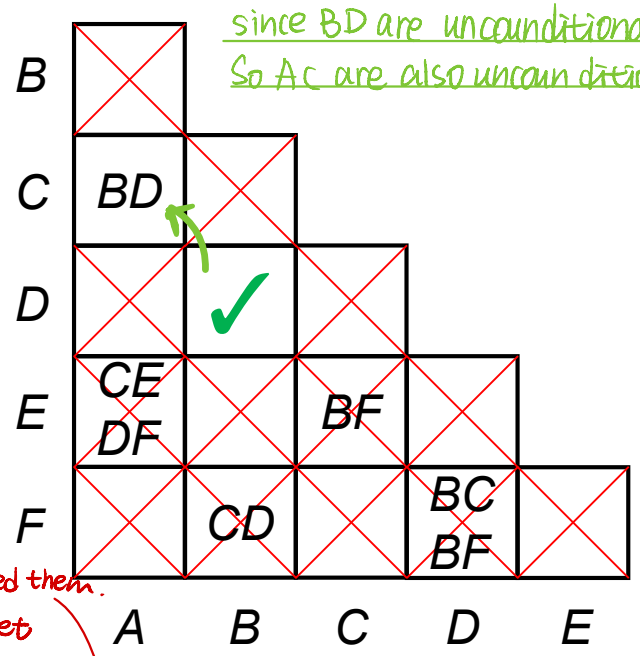
FF

If BD equivalent then BD equivalent?
Actually means: they are unconditionally equivalent.

Detecting Equivalent States Merger Table

PS	NS, z	
	$x = 0$	$x = 1$
A	E,0	D,1
B	F,0	D,0
C	E,0	B,1
D	F,0	B,0
E	C,0	F,1
F	B,0	C,0

since BD are unconditionally equivalent, So AC are also unconditionally equivalent

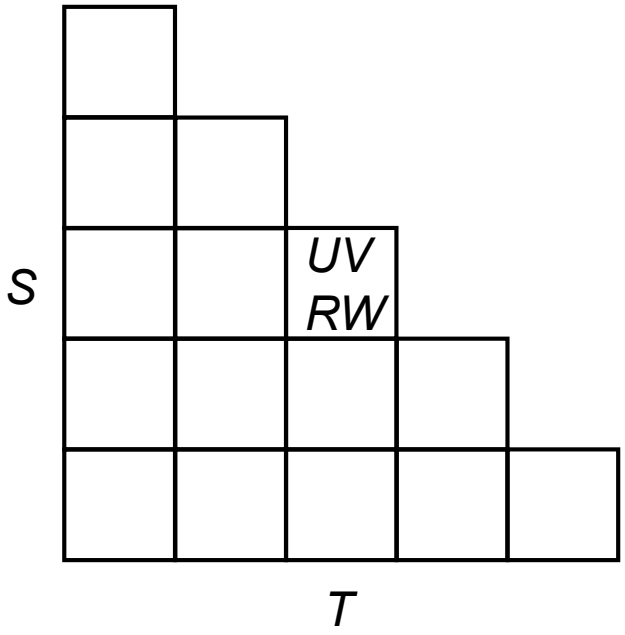


We take our set of states and we partitioned them.

~~A partition over set~~: is a set of subsets that covered by the entired set, and they have no common element.

{(AC),(BD),(E),(F)}

Equivalence Constraints



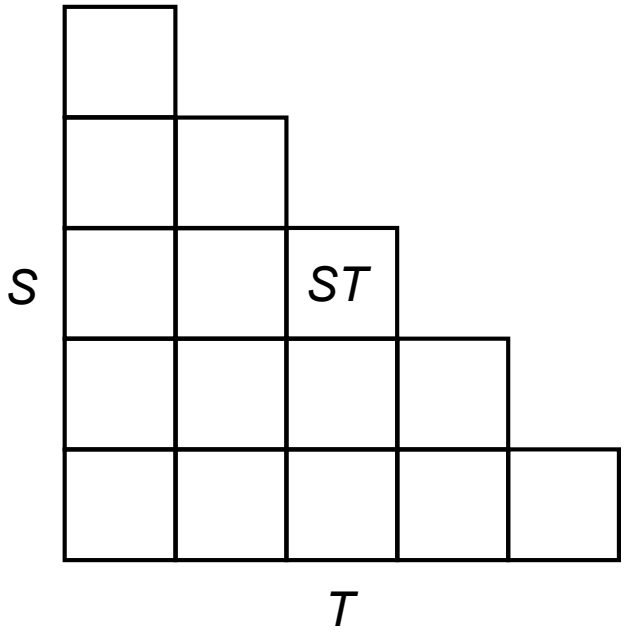
If S and T are equivalent,
then all of their next state pairs must be equivalent.

$$(S \equiv T) \rightarrow (U \equiv V) \cdot (R \equiv W)$$

If any pair of next states of S and T is not equivalent
then S and T are not equivalent

$$(U \equiv V)' + (R \equiv W)' \rightarrow (S \equiv T)'$$

Equivalence Constraints



If the next state pair for S and T is S and T
then S and T are unconditionally equivalent

$$(S \equiv T)' \rightarrow (S \equiv T)' = (S \equiv T) + (S \equiv T)' = 1$$

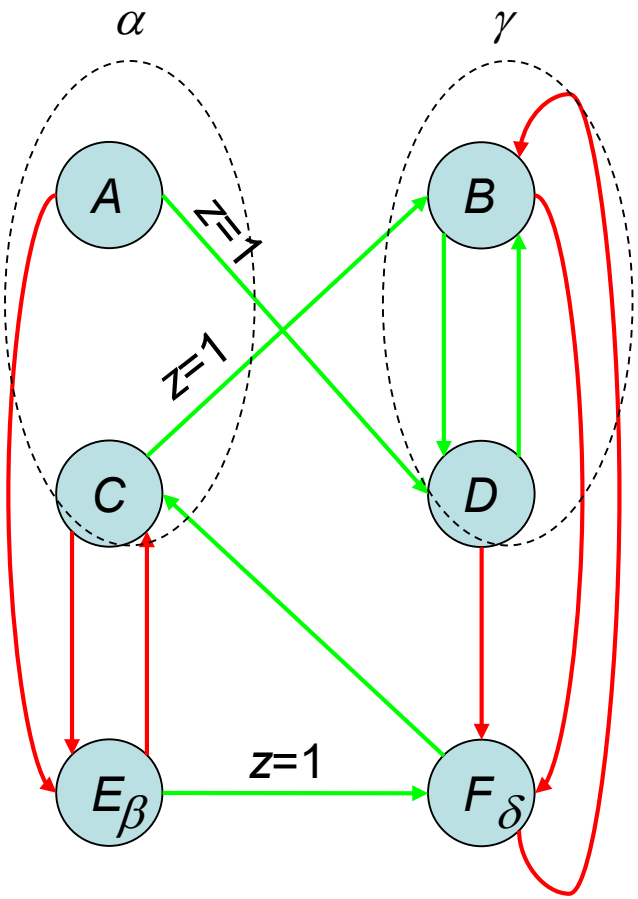
Reduced State Table

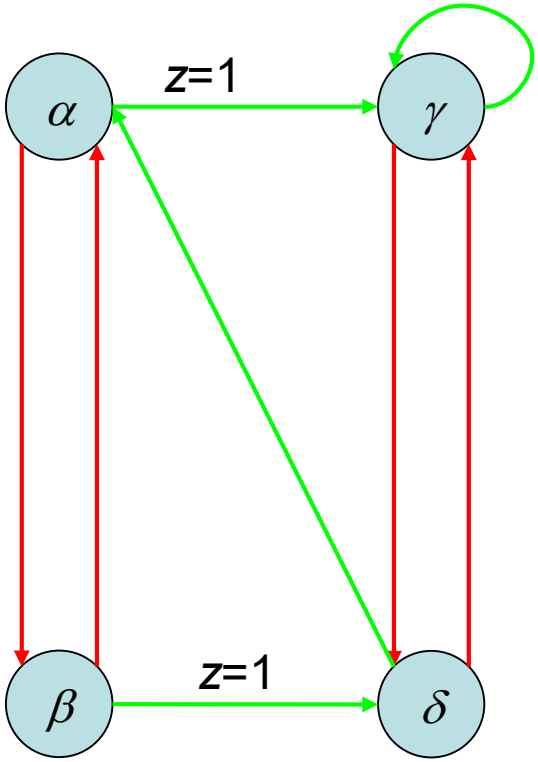
Equivalence Partition: $P = \{(AC), (E), (BD), (F)\}$

PS	NS, z	
	$x = 0$	$x = 1$
A	E,0	D,1
B	F,0	D,0
C	E,0	B,1
D	F,0	B,0
E	C,0	F,1
F	B,0	C,0

PS	NS, z	
	$x = 0$	$x = 1$
$\alpha = (AC)$	β, 0	γ, 1
$\beta = (E)$	α, 0	δ, 1
$\gamma = (BD)$	δ, 0	γ, 0
$\delta = (F)$	γ, 0	α, 0

Reduced State Diagram

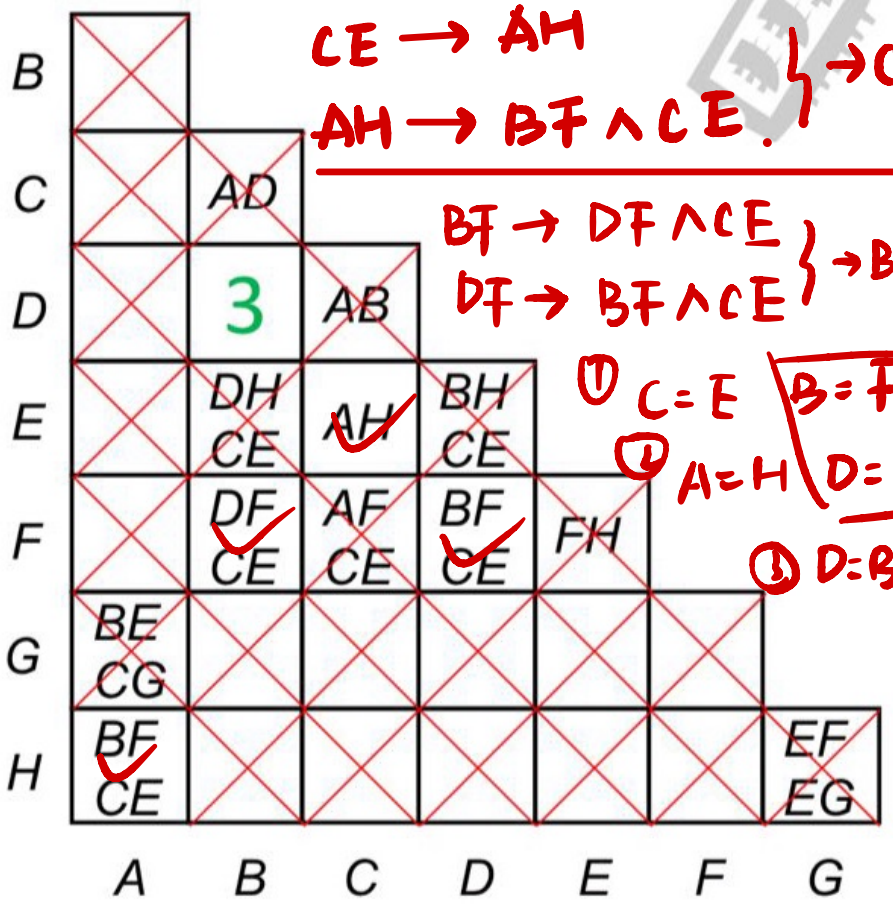




In-Class Exercise

PS	NS, z	
	x = 0	x = 1
A	B,0	C,1
B	D,1	C,0
C	A,1	C,0
D	B,1	C,0
E	H,1	E,0
F	F,1	E,0
G	E,0	G,1
H	F,0	E,1

(CE) (AH) (BF) (DF) (BD) BDF



CE → AH
AH → BF ∧ CE. } → CE

BF → DF ∧ CE
DF → BF ∧ CE } → BF

① C=E
② A=H
B=F D=F
③ D=B=F

(AH BF CE)(BF DF CE)(CE AH)(DF BF

AH BF CE DF AH BF CE DF

BF CE DF CE AH BF CE

{BF CE DF AH BD G}

{(AH), (BDF), (CE), G.}

In-Class Exercise

PS	NS, z	
	$x = 0$	$x = 1$
A	B,0	C,1
B	D,1	C,0
C	A,1	C,0
D	B,1	C,0
E	H,1	E,0
F	F,1	E,0
G	E,0	G,1
H	F,0	E,1